EMI REGISTRY MANUAL

EMIR Team

Document Version:	1.0.2
Component Version:	1.1.0
Date:	29 02 2012

This work is co-funded by the EC EMI project under the FP7 Collaborative Projects Grant Agreement Nr. INFSO-RI-261611.

Contents

1 Overview

The EMI Registry is a federated service registry based on REST-ful interfaces. The major functionalities of the registry includes

2 Features

The registry provides following features:

- the service registration includes the management of the services' information.
- Powerful data back-end based on MongoDB
- Schema-free information model based on JSON (using GLUE2 entity names for specific attributes)
- REST-ful API to browse the service registrations
- Security
  - PKI governed authentication
  - Distinguished Name based authorization using XACML driven PDP

For more information about EMI visit http://www.eu-emi.eu.

3 Introduction

The EMI Registry (EMIR) enables Grid services to be discovered in a DCI or a sub-domain of that DCI. It does not provide a full description of that service, only the minimal set of information that is required to find more details about that service. In this sense it acts as a bootstrapping mechanism for the DCI. The EMIR has considered the concept of DCI management, for example, only services or sub-domains that are part of the DCI would be discovered. This implies that the EMIR enables implementation of the DCI's management policies. As DCIs are distributed and based on federated sub-domains, the EMI Registry must also reflect such a model.

4 REST API

The EMI Registry allows Services to register/publish their capabilities while the Service Consumers are able to find the deployed services.

This section contains the description of the REST-ful interface, that allows the management of the service information (or entries) by exposing the individual URIs. The normative description of the API cab also be defined as Web Application Description Language (WADL) document WADL Section 9.

4.1 Register new Services

`HTTP Method` : POST

`URI` : /serviceadmin

`Content Type` : application/json

`Security Implications` : Requires authenticated "and" authorized user to perform this operation

4.1.1 Request

The message must contain a JSON document, includes an array of the JSON objects (see below), each of which is a service entry in the EMI registry.

Service description is defined as a Section 8 document.

Important
The only mandatory attribute is **Service_Endpoint_URL**, which should be unique

4.1.2 Response

The response contains similar array of JSON Objects as it was in sent request, confirming the successful update.

`Status Code` : OK / 200

4.2 Updating the Service information

`HTTP Method` : PUT

`URI` : /serviceadmin

`Content Type` : application/json

`Security Implications` : Requires an authenticated "and" authorized user to perform this operation

4.2.1 Request

The request body contain a similar JSON array object as defined POST method that contains the description of the Services to be updated. The Service Entries identified by the *Service_Endpoint_URL* key in the individual JSON objects will be updated respectively.

4.2.2 Response

The response contains similar array of JSON Objects as it was in sent request, confirming the successful update.

`Status Code` : OK / 200

4.3 Delete existing Services

`HTTP Method` : DELETE

`URI` : /serviceadmin

`Security Implications` : Requires an authenticated "and" authorized user to perform this operation

4.3.1 Request

The Service Entry identified by the URL will be deleted from the database if the client is properly authorized and the method were allowed by the security plugins.

`Query Parameters` : Service_Endpoint_URL= <*service unique URL*>

Example : /serviceadmin?Service_Endpoint_URL=http://1

4.3.2 Response

`Status Code` : OK / 200

4.4 Querying the EMI Registry database

`HTTP Method` : GET

`URI` : /services/query

`Content Type` : application/json

4.4.1 Request

The request contains the key-value pairs separated by ampersand `&`

`Query Parameters` : AttributeName=<Attribute_Value>&AttributeName=<Attribute_Value>&...

Example : /services/query?Service_Type=eu.emi.es&Service_Endpoint_HealthState=ok

The additional parameters can also be added to restrict and/or paginate the result

`Additional Query Parameters`:

```
skip=Integer value
```

skip returns the result skipping the given number of entries

```
limit=Integer value
```

limit defines the maximum number of result containing the service entries

4.4.2 Response

The response contains an array of service entries packed in a JSON array object

`Status Code`: OK / 200

4.5 Querying the EMI Registry database GLUE 2.0 XML format

`HTTP Method`: GET

`URI` : /services/query.xml

`Content Type` : application/xml

4.5.1 Request

The request contains the key-value pairs separated by ampersand `&`

`Query Parameters`: AttributeName=<Attribute_Value>&AttributeName=<Attribute_Value>&...

Example : /services/query?Service_Type=eu.emi.es&Service_Endpoint_HealthState=ok

The additional parameters can also be added to restrict and/or paginate the result

`Additional Query Parameters`:

```
skip=Integer value
```

skip returns the result skipping the given number of entries

```
limit=Integer value
```

limit defines the maximum number of result containing the service entries

4.5.2 Response

The response contains an XML document containing service entries in GLUE 2.0 format

`Status Code` : OK / 200

4.6 Viewing the Service information template

This To view the GLUE 2.0's JSON flavored service model.

`HTTP Method` : GET

`URI` : /model

`Content Type` : application/json

4.6.1 Request

N/A

4.6.2 Response

JSON document, as described in the `/serviceadmin` POST method

`Status Code` : OK / 200

4.7 Monitoring the Registry

Allows registry users to view the registry status

`HTTP Method` : GET

`URI` : /ping

4.7.1 Request

N/A

4.7.2 Response

`Status Code` : OK / 200

5 Installation

In order to install EMIR, it is required to install the SUN or OpenJDK Java 6 (JRE or SDK). If not installed on your system, you can download it from http://java.oracle.com

- Linux based operating system
- MongoDB[www.mongodb.org]

EMIR is distributed in the following formats:

- Platform independent format, provided in "tar.gz" format
- RPM package, suitable SL5 and other Fedora based Linux derivatives (RedHat, CentOS etc. . .)

IMPORTANT NOTE ON PATHS
The location of the installation and configuration files differ depending on type of bundle (see the above section).
If RPM bundle is chosen, the following paths will be used:

```
CONF=/etc/emi/emir
BIN=/usr/sbin
LOG=/var/log/emi/emir
```

The platform independent binary bundles all the files under single directory. Thus if extracted:

```
CONF=INST/conf
BIN=INST/bin
LOG=INST/logs
```

The above variables (CONF, BIN and LOG) will be used throughout the rest of this manual.

5.1 Installation from the self-contained archive (tar.gz)

Download the tar.gz archive from the EMIR website. The archive contains all the necessary files which are bundled already, therefore no special actions will be required except the exraction to final folder.

5.2 Installation using the RPM bundle

RPM Installation instructions.

6 Configuration

The EMIR server comes with a configuration file, a well documented file containing a number of options (security, database, multi-node setup etc…). The options are well defined and can be sufficient to setup and start-up the server in non-production environment, however the administrator needs to review before deploying on the production environment (DCI).

6.1 Server Configuration

The server configuration options are:

- Server host, port and scheme (plain or SSL)
- Settings of the type of the registry node, i.e. whether the current registry instance is a child of some other (a parent) regitsry node or a global registry.

6.2 MongoDB Configuration

The registry uses MongoDB to store and index the service records. Thus it is necessary to setup the Database before starting the EMIR.

—

—

6.3 Logging Configuration

The EMIR server provides log facilities to record all but some of the server activities. In order to change the logging configuration, CONF/log4j.properties should be reviewed by the administrator.

—

—

6.4 Authentication with X.509

The authentication of the client in EMIR is based on PKI, thus using X.509 standard is used to authenticate the client as well as the server. However this implies the EMIR client and the server should own a X.509 certificate issued by a mutually trusted certificate authority (CA).

—

—

6.5 Authorization

The EMIR offers two mutually exclusive options to authorize its clients.

- Using Access Control List (ACL)
- XACML based authorization

6.6 ACL Based Authorization

This is the standard mechanism in EMIR to authorize the clients. The incoming requests' subject is matched against the pre-defined ACL file (CONF/emir.acl). The ACL file has a list of DN/role pairs separated by **::**, see the example below

emir.acl

```
emailAddress=emiregistry@user.eu,CN=EMIRegistry-Demo-User,OU=JSC,O= ↩
    Forschungszentrum Juelich GmbH,L=Juelich,C=DE :: serviceowner
emailAddress=emiregistry@user.eu-admin,CN=EMIRegistry-Demo-User- ↩
    Admin,OU=JSC,O=Forschungszentrum Juelich GmbH,L=Juelich,C=DE :: ↩
     admin
```

the property in the `CONF/dsr.config` file

—

—

There are only two pre-defined roles in EMIR that can be selected: the *administrator* who can change any registration, while the *serviceowner* is only allowed to change (modify/delete) her already created registrations.

Important
The server should be running on SSL mode to anable the authorization

6.7 Fine Grained Authorization with XACML

Using XACML 2.0 is an alternative way to authorize the subjects in a fine grained manner. The administrator should review the policies defined in the `CONF/xacml2Policies/` folder and change them according to the infrastructure needs. However the already defined policies provides a good starting point to the administrators to define/modify the policies. In order to enable the given authorization following property must be set

Table 1: Settings of the Attribute Sources

—

—

Table 2: Access Control Settings

—

Table 2: (continued)

7 EMIR Client Daemon (EMIRD)

7.1 About the EMIRD

The UMD services need to be registered into the EMI Registry service infrastructure to be discoverable for the clients. Most of the services or even the containers executing them provide a way to do this but not all of them. For those that are unable to register themselves automatically and periodically the EMIRD is available.

The EMIRD is a daemon like service that can be executed next to these services (preferably on the same machine) and able to perform the automatical and periodical registration and update against the configured EMI Registry service instead of the service itself. This client uses exactly the same, standard RESTful API as the other clients do.

Most of the parameters of these registrations and updates can be configured. For the details see the Configuration section!

After the successful registration until the termination of the daemon, the EMIRD client do the periodical updates then finally, when the execution of the daemon is over, it attempts to delete the service entries from the remote database.

The service entries can be simple or advanced ones.

The simple service entries contain only the mandatory and easily configurable attributes that are the following: Service_Name, Service_Type, Service_Endpoint_URL, Service_Endpoint_InterfaceName. Here, the single mandatory element is the Service_Endpoint_URL.

The advanced entries can contains any kind of key value pairs that are accepted by the EMI Regisrty services and can be configured in the form of whole, formatted **json** documents.

7.2 Installation

The installation of the EMIRD client is trivial. The only thing to do is to install the `emir-daemon` package from the EMI repository by executing:

```
yum instlal emir-daemon
```

7.3 Configuration

The configuration of EMIRD can be performed by editing its configuration file or files. The configuration can be found basically in one file that default location is `/etc/emi/emird/emird.ini`.

This file contains every configuration options that can be the EMIRD daemon control by, like *service url*, *logging verbosity*, *credential location*, etc.

The advanced service entries to be propagated can be described in separated configuration files preferably also under this directory and use to have `.json` extension.

The main configuration file has INI format. The `emir` section contains the daemon scoped options while the others are to describe the different service entries to be registered. In these cases the exact name is indifferent, they just have to differ from eachother and must avoid the `emir` name as well.

7.3.1 Configuration options

Note: The option names are case-insensitives.

url

`Location:` `emir` section

`Default value:` *No default value*

`Mandatory:` Yes

`Description:`

URL of the EMIR service to connect in a protocol://domain:port format.

The protocol part is not mandatory if https (default) The port part is not mandatory if 54321 (default) The domain part is mandatory

Examples: url = emiregistry2.grid.niif.hu url = https://emiregistry2.grid.niif.hu url = https://emiregistry2.grid.niif.hu:54321

period

`Location:` `emir` section

`Default value:` *No default value*

`Mandatory:` Yes

`Description:`

The period of the registration/update messages. Its value is given in minutes.

validity

`Location:` `emir` section

`Default value:` *No default value*

`Mandatory:` Yes

`Description:`

The validity of the registration entries. Its value is given in minutes.

cert

`Location:` `emir` section

`Default value:` /etc/grid-security/hostcert.pem

`Mandatory:` No

`Description:`

User certificate file location in PEM format. Only used and checked if the protocol in the url option is *https*.

key

`Location:` `emir` section

`Default value:` /etc/grid-security/hostkey.pem

`Mandatory:` No

`Description:`

User key file location in PEM format. Only used and checked if the protocol in the url option is *https*.

cadir

`Location:` `emir` section

`Default value:` /etc/grid-security/certificates

`Mandatory:` No

`Description:`

A path pointing to the store where the PEM certificate of the trusted Certificate Authorities can be found. Only used and checked if the protocol in the url option is *https*.

verbosity

`Location`: `emir` section

`Default value`: error

`Mandatory`: No

`Description`:

Logging verbosity. The parameter is optional. If missing or an invalid value is given, the default value will be used. The logs are written into the log file that can be found in the */var/log/emi/emird* directory by default.

Service_Endpoint_URL

`Location`: simple service entry section

`Default value`: *No default value*

`Mandatory`: Yes

`Description`:

The Service Endpoint URL to be propagated. If this option is missing an error message will be raised.

Service_Name

`Location`: simple service entry section

`Default value`: *No default value*

`Mandatory`: No

`Description`:

The Service Name to be propagated. If this option is missing then the service entry will contains no such component.

Service_Type

`Location`: simple service entry section

`Default value`: *No default value*

`Mandatory`: No

`Description`:

The Service Type to be propagated. If this option is missing then the service entry will contains no such component.

Service_Endpoint_InterfaceName

`Location`: simple service entry section

`Default value`: *No default value*

`Mandatory`: No

`Description`:

The Service Endpoint Interface Name to be propagated. If this option is missing then the service entry will contains no such component.

json_file_location

`Location`: advanced service entry section

`Default value`: *No default value*

`Mandatory`: Yes

`Description`:

The service entry can be also defined in a single external json formatted file per service. Any allowed json attributes are allowed in this way. The location of this file must be defined in this ini variable. The recommended place for these files is under the `/etc/emi/emird/` directory and naming them after the name of the given service with `.json` extension.

8 Appendix I

The service record JSON template of EMIR interface.

```
[
        {
                "Service_ID":"s1",
                "Service_Name":"ComputingService",
                "Service_CreationTime":{"$date":"2011-07-21T11 ↩
                    :47:24Z"},
                "Service_Type":"job-management",
                "Service_Contact": [{"ContactType":"sysadmin", " ↩
                    Detail":"http://contactlink"},{"ContactType":" ↩
                    developer", "Detail":"http://contactlink"}],
                "Service_Capability":["capability1","capability2"],
                "Service_QualityLevel":"production",
                "Service_Complexity":"complexity",
                "Service_Validity": 12313,
                "Service_Extensions":[{"key":"value"},{"key":"value ↩
                    "}],
                "Service_Endpoint_ID":"se1",
                "Service_Endpoint_URL":"http://1",
                "Service_Endpoint_Capability":["capability1"," ↩
                    capability2"],
                "Service_Endpoint_Technology":"technology",
                "Service_Endpoint_InterfaceName":"interface",
                "Service_Endpoint_InterfaceVersion":["version1"," ↩
                    version2"],
                "Service_Endpoint_InterfaceExtension":["extension1 ↩
                    ","extension2"],
                "Service_Endpoint_WSDL":"http//1.wsdl",
                "Service_Endpoint_SupportedProfile":["profile1"," ↩
                    profile2"],
                "Service_Endpoint_Semantics":["semantic1"," ↩
                    semantic2"],
                "Service_Endpoint_HealthState":"ok",
                "Service_Endpoint_HealthStateInfo":"state info",
                "Service_Endpoint_ServingState":"production",
                "Service_Endpoint_StartTime":{"$date":"2011-07-21 ↩
                    T11:47:24Z"},
                "Service_Endpoint_IssuerCA":"issuer-dn",
                "Service_Endpoint_TrustedCA":["dn1","dn2","dn3"],
                "Service_Endpoint_DowntimeAnnounce":{"$date ↩
                    ":"2011-07-21T11:47:24Z"},
                "Service_Endpoint_DowntimeStart":{"$date ↩
                    ":"2011-07-21T11:47:24Z"},
                "Service_Endpoint_DowntimeEnd":{"$date":"2011-07-21 ↩
                    T11:47:24Z"},
                "Service_Endpoint_QualityLevel":"production",
                "Service_Location_Address":"A Street 1",
                "Service_Location_Place":"Bonn",
                "Service_Location_Country":"Germany",
                "Service_Location_PostCode":"53119",
                "Service_Location_Latitude":53.3,
                "Service_Location_Longitude":4,
                "Service_ExpireOn":{"$date":"2011-07-21T11:47:24Z"}

        },
        {
                "Service_ID":"s2",
                "Service_Name":"ComputingService",
                "Service_CreationTime":{"$date":"2011-07-21T11 ↩
                    :47:24Z"},
                "Service_Type":"job-management",
                "Service_Contact": [{"ContactType":"sysadmin", " ↩
                    Detail":"http://contactlink"},{"ContactType":" ↩
                    developer", "Detail":"http://contactlink"}],
                "Service_Capability":["capability1","capability2"],
                "Service_QualityLevel":"production",
                "Service_Complexity":"complexity",
                "Service_Validity": 12313,
                "Service_Extensions":[{"key":"value"},{"key":"value ↩
                "}],
                "Service_Endpoint_ID":"se2",
                "Service_Endpoint_URL":"http://2",
                "Service_Endpoint_Capability":["capability1"," ↩
                    capability2"],
                "Service_Endpoint_Technology":"technology",
                "Service_Endpoint_InterfaceName":"interface",
                "Service_Endpoint_InterfaceVersion":["version1"," ↩
                    version2"],
                "Service_Endpoint_InterfaceExtension":["extension1 ↩
                    ","extension2"],
                "Service_Endpoint_WSDL":"http//1.wsdl",
                "Service_Endpoint_SupportedProfile":["profile1"," ↩
                    profile2"],
                "Service_Endpoint_Semantics":["semantic1"," ↩
                    semantic2"],
                "Service_Endpoint_HealthState":"ok",
                "Service_Endpoint_HealthStateInfo":"state info",
                "Service_Endpoint_ServingState":"production",
                "Service_Endpoint_StartTime":{"$date":"2011-07-21 ↩
                    T11:47:24Z"},
                "Service_Endpoint_IssuerCA":"issuer-dn",
                "Service_Endpoint_TrustedCA":["dn1","dn2","dn3"],
                "Service_Endpoint_DowntimeAnnounce":{"$date ↩
                    ":"2011-07-21T11:47:24Z"},
                "Service_Endpoint_DowntimeStart":{"$date ↩
                    ":"2011-07-21T11:47:24Z"},
                "Service_Endpoint_DowntimeEnd":{"$date":"2011-07-21 ↩
                    T11:47:24Z"},
                "Service_Endpoint_QualityLevel":"production",
                "Service_Location_Address":"A Street 1",
                "Service_Location_Place":"Bonn",
                "Service_Location_Country":"Germany",
                "Service_Location_PostCode":"53119",
                "Service_Location_Latitude":53.3,
                "Service_Location_Longitude":4,
                "Service_ExpireOn":{"$date":"2011-07-21T11:47:24Z"}
        },

]
```

9 Appendix II

The EMIR WADL document to define the REST-ful API

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://wadl.dev.java.net/2009/02">
```

```
    <doc xmlns:jersey="http://jersey.java.net/" jersey:generatedBy ↩
        ="Jersey: 1.9.1 09/14/2011 02:05 PM"/>
    <grammars/>
    <resources base="https://localhost:54321/">
        <resource path="/children">
            <method id="childDSRs" name="GET">
                <response>
                    <representation mediaType="*/*"/>
                </response>
            </method>
            <method id="checkin" name="POST">
                <response>
                    <representation mediaType="*/*"/>
                </response>
            </method>
        </resource>
        <resource path="/neighbors">
            <method id="childDSRs" name="GET">
                <response>
                    <representation mediaType="*/*"/>
                </response>
            </method>
        </resource>
        <resource path="/parent">
            <method id="childDSRs" name="GET">
                <response>
                    <representation mediaType="*/*"/>
                </response>
            </method>
        </resource>
        <resource path="/serviceadmin">
            <method id="getServicebyUrl" name="GET">
                <response>
                    <representation mediaType="application/json"/>
                </response>
            </method>
            <method id="registerServices" name="POST">
                <request>
                    <representation mediaType="application/json"/>
                </request>
                <response>
                    <representation mediaType="application/json"/>
                </response>
            </method>
            <method id="updateServices" name="PUT">
                <request>
                    <representation mediaType="application/json"/>
                </request>
                <response>
                    <representation mediaType="application/json"/>
                </response>
            </method>
            <method id="deleteService" name="DELETE">
                <response>
                    <representation mediaType="*/*"/>
                </response>
            </method>
        </resource>
        <resource path="/services">
            <method id="queryWithParams" name="GET">
                <response>
                    <representation mediaType="application/json"/>
                </response>
            </method>
            <method id="queryWithJSON" name="POST">
                <request>
                    <representation mediaType="application/json"/>
                </request>
                <response>
                    <representation mediaType="application/json"/>
                </response>
            </method>
            <method id="queryXMLWithJSON" name="POST">
                <request>
                    <representation mediaType="application/json"/>
                </request>
                <response>
                    <representation mediaType="application/xml"/>
                </response>
            </method>
            <method id="queryXMLWithParams" name="GET">
                <response>
                    <representation mediaType="application/xml"/>
                    <representation mediaType="text/xml"/>
                </response>
            </method>
            <resource path="/urls">
                <method id="getServiceEndPoints" name="GET">
                    <response>
                        <representation mediaType="application/json ↩
                            "/>
                    </response>
                </method>
            </resource>
            <resource path="/types">
                <method id="getServiceTypes" name="GET">
                    <response>
                        <representation mediaType="application/json ↩
                            "/>
                    </response>
                </method>
            </resource>
            <resource path="/query.xml">
                <method id="queryXml" name="GET">
                    <response>
                        <representation mediaType="application/xml ↩
                            "/>
                        <representation mediaType="text/xml"/>
                    </response>
                </method>
            </resource>
            <resource path="/pagedquery">
                <method id="pagedQuery" name="GET">
                    <response>
                        <representation mediaType="*/*"/>
                    </response>
                </method>
            </resource>
        </resource>
        <resource path="/model">
            <method id="getModel" name="GET">
                <response>
                    <representation mediaType="text/html"/>
                    <representation mediaType="application/json"/>
                </response>
            </method>
        </resource>
        <resource path="/ping">
            <method id="ping" name="GET">
                <response>
                    <representation mediaType="application/json"/>
                    <representation mediaType="text/plain"/>
                </response>
            </method>
        </resource>
    </resources>
</application>
```